WEST

Generate Collection | Print

L10: Entry 5 of 6 File: USPT Jan 11, 2000

DOCUMENT-IDENTIFIER: US 6014673 A
TITLE: Simultaneous use of database and durable store in work flow and process flow systems

Abstract Text (1):
A method and apparatus for reliable high-speed access to a database system that stores system data in a non-volatile database, stores current data in an online database object cache, the current data reflecting at least a portion of the system data in the non-volatile database and the online database object cache providing the database system with the capability of querying and updating the current data in the online database object cache, logging each message in the database system as an entry in a durable log file, and periodically committing the current data in the online database object cache to the non-volatile database.

Brief Summary Text (2):
This invention relates to a method and apparatus for implementing simultaneous usage of a database and durable storage of database updates and more particularly to simultaneous usage of a database and durable storage for a workflow and process flow management system.

Brief Summary Text (5):
Database systems provide the needed reliability, but at substantial performance cost. To assure maximal reliability using a database management system, all updates must be durable (permanent) before acting upon them. This generally requires that the database updates be committed.

Brief Summary Text (6):
A conventional database system maintains the data in a durable storage mechanism, such as a disk drive. The database system will also typically have a non-durable copy of an active portion of the database in a volatile memory cache. The data in volatile memory can be rapidly accessed, but can be destroyed and lost in the event of a system crash, program failure or similar abnormal termination. To maintain the integrity of the database, updates to the database must be guaranteed to be stored, i.e. committed, in the durable storage mechanism. A commit requires that the database system store all modified data in memory cache to the durable storage mechanism.

Brief Summary Text (7):
However, durable storage requires more access time than cache memory. A process that requests the database system to commit an update transaction must wait for an acknowledgment that the commit was successful. The database system only acknowledges the commit when the data has been updated in durable storage. Frequent commits, therefore, degrade system performance because of the time consumed by processes waiting for acknowledgment from the database system.

Brief Summary Text (8):
A database system can be made reliable by storing all state changes in a durable log file. However, the modified data, as represented by the state changes in the log file, would not be easily accessible and much of the utility of the database system is lost.

Brief Summary Text (16):
In addition to the data above, it is useful for some workflow process applications to have access to historical data regarding state changes within the system. Historical data takes the form of an audit trail for completed workflow processes and is useful to the collection of statistical data for process and resource bottleneck analysis, flow optimization and automatic workload balancing. While workflow systems need full ACID properties over some of their data, they do not need them for the historical data. The

existing workflow systems that use databases either record historical data along with current data in their database, discard the historical data entirely, or store it non-durably.

Brief Summary Text (18):
It is therefore desirable to provide a high degree of reliability in a database by durably storing updated data and to simultaneously provide rapid access to the database. It is particularly desirable in workflow systems to not lose data modifications related to work assignments. It is also desirable in a workflow system to durably store the historical data of the system and provide convenient access to the data.

Brief Summary Text (20):
The present invention is a database system and durable store method that partitions information into information which needs to be up-to-date online in the database, information which can be brought up-to-date offline in the database, and information which need never be stored into the database, thereby providing substantially guaranteed reliability for the information. All updates are durably stored before being acted upon to provide maximal reliability. Such updates are stored by recording the message traffic through a database engine in a log file whereby all updates originate from such a message. Database changes need not be durably recorded immediately to avoid incurring a penalty. Offline procedures store the historical data from the log file into a relational database allowing the historical data to be queried.

Brief Summary Text (21):
The online use of the log file eliminates the requirement that the system commit all database changes before acting upon any changed state. This dramatically increases the throughput of the system. The offline relational storage of historical data allows users convenient and efficient access to the data. In the event of a system failure, all externally knowable states can be recovered whereby all the system states upon which any action has been performed is restored.

Brief Summary Text (22):
One feature of the present invention is in the application of the "write to log first" principle to provide reliability to information not being directly committed into a database. Another feature of the present invention is durable logging of state changes without requiring online database commits. Yet another feature of the present invention is the use of an offline relational database for historical purposes. And still another feature of the present invention is system recovery which guarantees no knowable state is lost.

Detailed Description Text (15):
A work node 41 is a placeholder for a process activity, which is a logical representation of a piece of work contributing towards the accomplishment of a process. A process activity is mapped to the invocation of an operation on business objects during the execution of the process and each process activity can represent a manual operation by a human or a computerizable task to execute legacy applications 30, 31, 32, 33 (shown in FIG. 1), access application databases 34a, 34b (also shown in FIG. 1), control instrumentation, sense events in the external world or effect physical changes. A process activity definition includes a forward activity and optionally, a compensation activity, a cancel activity, a resource management activity, timeout and deadline information and input and output data.

Detailed Description Text (18):
Rule nodes 42, 44, 47, 49, 51, 53, 55 are executed each time any inward arc fires. Work nodes 41, 43, 45, 46, 48, 50, 52, 54 have states of initial or fired. When the inward arc is fired on a work node 41 in the initial state, the work node 41 changes its state to fired and performs or requests its associated activity. When the inward arc is fired on a work node 41 in the fired state, nothing is done.

Detailed Description Text (19):
A reset arc, for example, between nodes 42-45, together with the forward arcs between its destination and source, forms a loop. When traversed, a reset arc causes all nodes 42-45 within its loop to be reset. Resetting a fired work node 43 changes its state to initial so that the node 43 can be re-executed. Resetting an active work node 43 cancels the current execution of the corresponding process activity and changes its state to initial.

Detailed Description Text (28):

FIGS. 6, 7A, 7B, 7C, 7D and 8 illustrate an example of a series of messages between engine 20 and business objects 93a and 93b. As described above, engine 20 creates Msg1 for transmission to business object 93a. FIG. 7A the destites an example of the destination address list 154 identified by the destination address list 154 contained in Msg1. Destination address list 154 identifies business object 93a as the first object on the list which prompts engine 20 to dispatch Msg1 to business object 93a. Engine 20 places itself on the source address list 156. Msg1 is first logged as an entry in log file 76, then engine 20 transmit Msg1 to business object 93a. Msg 3 and Msg 4 are similarly dispatched and logged as shown in FIGS. 7C and 7D respectively.

Detailed Description Text (33):
Application Data Handlers, such as data handlers 92a-92c, provide for the provision of ASData to a Business Object and the storage of updates to ASData at the conclusion of the Business Object's execution.

Detailed Description Text (41):
ASData must be maintained in non-volatile mass storage in database 21 so that it is preserved in the event of a crash. However, as noted above, updates to ASData during the course of workflow processing would present a substantial performance penalty if the engine 20 is required to commit the updates in each transaction to avoid losing data in the event of a crash.

Detailed Description Text (43):
The architecture of the present invention addresses these problems by using an online database object cache 75 to store current data, which is a volatile version of ASData that reflects recent updates to the ASData in database 21, the durable log file 76 that contains messages representing historical data, and non-volatile storage in database 21.

Detailed Description Text (45):
In addition, the present invention permits database manager 64 to more efficiently schedule disk I/O operations for non-volatile database 21. When a user process updates data in object cache 75, the user process may continue processing without waiting for a commit reply from the database manager 64. Database manager 64 is then free to independently schedule the write of the modified data to the disk containing non-volatile database 21. This permits disk I/O operations for database 21 to be interspersed with the disk operations for other system functions such as computational I/O or network traffic I/O. Database updates that are written to disk do not have to be rewritten to disk when performing a periodic database commit.

Detailed Description Text (47):
Entries in log file 76 each contain a backward pointer to the previous record for the same process instance. In the event of restart after system failure, the OpenPM engine 20 replays any entries in log file 76 that are not reflected in the database 21. This provides the reliability that other workflow systems only achieve by committing after each database change but with greater system throughput.

Detailed Description Text (49):
Each message inbound to the OpenPM engine 20 is logged in log file 76 and then queued for appropriate processing by the engine. All ASData updates are stored in object cache 75 and logged in log file 76. Each response from engine 20 is also logged. Periodically database manager 64 commits the updates to non-volatile database 21 and logs a Commit point message in log file 76. Because the database changes are periodically committed in a batch, it is not necessary to commit each database transaction.

Detailed Description Text (50):
A database commit can take several forms in the present invention depending on how the database manager 64 is implemented. In one embodiment, database manager 64 schedules a write to disk where non-volatile database 21 resides whenever ASData is updated. The database commit then checks the log file for entries that have not been written to disk, performs the disk write for any entries found, and logs a commit point.

Detailed Description Text (51):
The database manager 64 may also be implemented where all disk writes are performed at commit time. In this case, database manager 64 performs a disk write of each transaction entry recorded in the log file since the last commit point and then logs a commit point. Yet another approach is for database manager 64 to mark the ASData in object cache 75 as updated, in addition to making a log entry, and then writing the updated ASData to non-volatile database 21 at commit time before logging a commit

point.

Detailed Description Text (54):
The present invention provides guaranteed at least once behavior. The engine 20 will strive to deliver each message at least once regardless of a failure of the engine. However, the engine may deliver a message more than once. The engine 20 may send replicated messages while restarting after a failure. These messages will faithfully replicate the message IDs of the original messages that they represent, although it is possible that the contents may differ.

Detailed Description Text (55):
The present invention will not replicate messages for which replies have been received and logged. Applications must either correctly tolerate duplicate messages with possibly differing contents or ensure that they do not occur.

Detailed Description Text (56):
The present invention is not capable of independently and robustly surviving certain situations. If the non-volatile database 21 or log file 76 suffer damage the system may not properly recover. Note that a crash before a database commit does not in itself constitute damage because the present invention can recover. Installations requiring robust behavior in the face of such possibilities must consider adding additional mechanisms such as mirrored discs, replicated databases, and the like.

Detailed Description Text (62):
The engine 20 then goes on to "Verify existence of database" 226 and "Verify proper updatability of database" 228. These steps determine that the database 21 is still intact and accessible to the engine 20. If either of these steps fail, then engine 20 writes a failure message to the log file 76 and then sends a message to the operator.

Detailed Description Text (65):
If the Validate phase 192 is successful, the engine 20 moves on to the Recover Database phase 194 shown in FIG. 11. First, the engine initializes all system queues from their values stored in the database 21. Next, the engine finds the last Commit point entry in log file 76. Starting from the last Commit point, the engine 20 rolls forward in log file 76 reexecuting the database updates recorded in the log file 76 in the current data in the object cache 75. The engine then queues all messages in the log file 76 for retransmittal in a resendQueue using the original message IDs. All intra-engine messages are then dequeued leaving only messages for transmittal to objects outside the engine. Then, the engine 20 dequeues any messages in resendQueue for which there is an entry for a response message in the log file 76. This process takes into account all messages in log file 76 until the Recovery.sub.-- Begin.sub.-- Phase.sub.-- 1 message is found in log file 76. Finally, the engine 20 commits the updates to the current data to database 21 and writes a Commit Point entry in the log.

Detailed Description Text (68):
Upon entering the Normal Processing phase 198, shown in FIG. 13, the engine 20 logs a Recovery.sub.-- End message in log file 76. The engine 20 then enters the normal operating state of the system wherein it periodically commits the updates to current data in object cache 75 to the database 21 and logs a Commit point message in log file 76.

Detailed Description Text (69):
The previous discussion showed how to perform recovery when all engine 20 queues reside in the non-volatile database 21 and the main problem is recreating any database updates not committed before a failure. The recovery algorithms can be extended to cover the case where some or all of the system processing queues are transient data not stored in database 21.

Current US Original Classification (1):
707/202

CLAIMS:

1. A method for reliable high-speed access to a database system, comprising:

storing system data in a non-volatile database;

storing current data in an object cache, the current data reflecting at least a portion of the system data in the non-volatile database, the object cache providing the

database system with the capability of querying and updating the current data; database and the object cache, wherein (1) each message contains historical and updating data of the current data and (2) the log file allows recovery of the database system in the event of a system failure by reconstructing the current data in the object cache, wherein no database commit is required in the log file or the database when the log file logs each message; and

periodically committing the current data in the object cache to the non-volatile database.

3. The method of claim 2, further comprising the step of marking current data in the object cache when the current data is updated.

9. A database system configured to support a workflow process, comprising:

a non-volatile database configured to reside in non-volatile storage to support a data structure of the workflow process and to provide the workflow process with access to a first plurality of data elements;

a volatile database configured to support the data structure of the workflow process and to provide the workflow process with query and update access to a second plurality of data elements, the second plurality of data elements being related to a portion of the first plurality of data elements;

a non-volatile and non-database log file in a non-volatile memory separate from the databases to log, as an entry, each message generated in the database system, wherein (1) each message contains historical and updating data of the second plurality of data elements and (2) the log file allows recovery of the database system in the event of a system failure by reconstructing the second plurality of data elements in the volatile database, wherein no database commit is required in the log file or the database when the log file logs each message; and

an engine configured to support the workflow process and to log an entry in the non-volatile log file for each message generated by the workflow process.

10. The database system of claim 9, wherein the engine is further configured to periodically commit the second plurality of data elements to the first plurality of elements by searching for entries in the non-volatile log file related to update transactions on the second plurality of data elements and reexecuting the entries on the first plurality of data elements.

12. The database system of claim 11, wherein the engine is further configured to recover from failure by locating a most recent commit entry in the non-volatile log file, to identify the entries in the non-volatile log file related to update transactions on the second plurality of data elements that are subsequent to the most recent commit entry, and to reexecute the identified entries on the first plurality of data elements.